ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21336 - 21335

LECTURERS : Prof. Dr. Mustafa Ersel Kamaşak

Prof. Dr. Deniz Turgay Altılar

GROUP MEMBERS:

150220704 : BEYZA TÜRK

150140709 : İLKE BAŞAK BAYDAR

SPRING 2024

Contents

1 INTRODUCTION

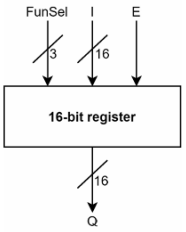
In this project, we have implemented registers, register file including Address Register, Instruction Register, Arithmetic Logic Unit and Arithmetic Logic Unit System. Moreover, inside this implementation, we can make arithmetic and logical operations and we can store the results of the operations with using memory and registers.

2 MATERIALS AND METHODS

To implement this project, we used Verilog Hardware Description Language. As source materials, we benefited the given Verilog introduction slides, Digital Circuit Course's slides [1] and the textbook Computer System Architecture by Morris Mano[2]. We carried out the implementation of this project in four parts. We designed, tested, simulated and reported each part together.

2.1 PART 1: 16-BIT REGISTER

For the first part, we designed and implemented 16-bit general register which is utilized afterwards in address register and instuction register and their files. This general register has 8 functionalities that are controlled by 3-bit control signals(FunSel), enable(E) inputs a clock signal. Register has 16-bit input(I) and 16-bit output(Q).(Figure 1) The schematic of General Register implementation is given below(Figure 2)



E	FunSel	Q^+
0	ϕ	Q (Retain value)
1	000	Q-1 (Decrement)
1	001	Q+1 (Increment)
1	010	I (Load)
1	011	0 (Clear)
1	100	Q (15-8) ← Clear, Q (7-0) ← I (7-0) (Write Low)
1	101	Q (7-0) ← I (7-0) (Only Write Low)
1	110	Q (15-8) ← I (7-0) (Only Write High)
1	111	Q (15-8) ← Sign Extend (I (7)) Q (7-0) ← I (7-0) (Write Low)

Figure 1: Graphic Symbol of the Register and the Characteristic Table

.

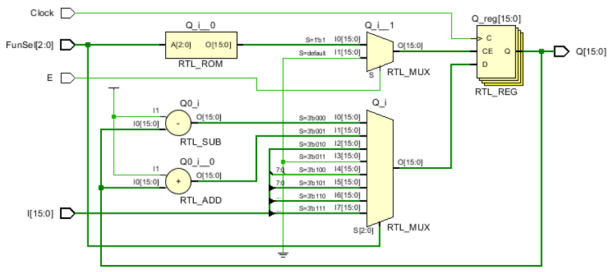


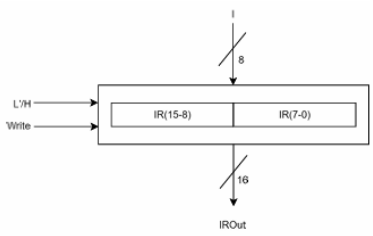
Figure 2: Verilog HDL Schematic of General Register

2.2 PART 2: REGISTER FILE

A register file is an array of processor registers situated within the central processing unit (CPU). It serves as a staging area for data transfer between memory and the functional units present on the chip. We designed a register file which is a structure that contains many registers. To implement a register file, we designed an Instruction Register(IR), a system which contains 8 different register and an Address Register(AR) by using general register in part 1.

2.2.1 Instruction Register:

Instruction register is a component within the central processing unit (CPU) of a computer or similar device. It stores programming instructions to be executed in the upcoming clock cycle, following directives from other components.[3] We designed a 16-bit IR register whose block diagram and function table are given in Figure 3. This register can store 16-bit binary data, the input of this register file is 8-bits. Therefore, we utilized the 8-bit input bus to load either the lower half (bits 7-0) or the upper half (bits 15-8) based on the L'H signal.



L'H	Write	IR⁺
ϕ	0	IR (retain value)
0	1	IR (7-0) ← I (Load LSB)
1	1	IR (15-8) ← I (Load MSB)

Figure 3: Graphic Symbol of the Instruction Register and the Characteristic Table

The schematic of IR implementation is given below(Figure 4).

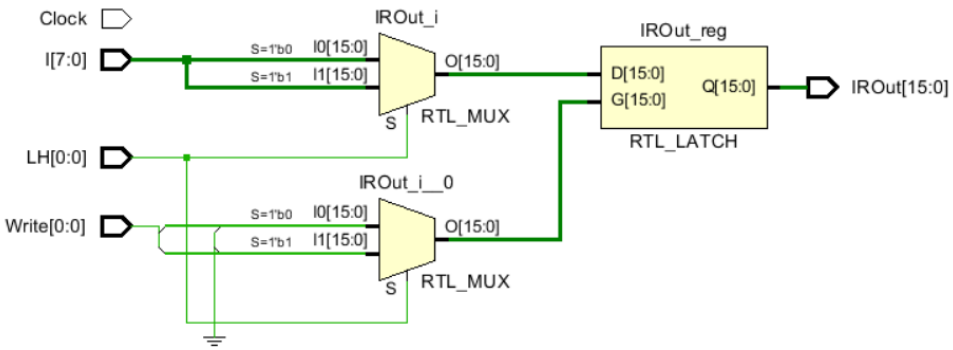


Figure 4: Verilog HDL Schematic of IR

2.2.2 8-Register System:

The depicted system in Figure 7 comprises four 16-bit general-purpose registers named R1, R2, R3, and R4, along with four 16-bit scratch registers labeled S1, S2, S3, and S4. The specifications for inputs and outputs are outlined as follows (Figure 5). OutASel and

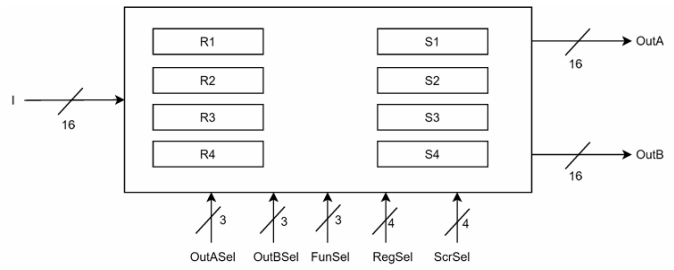


Figure 5: 16-bit general purpose and scratch registers, inputs, and outputs

OutBSel control inputs determine which registers are selected to feed data to the 16-bit output lines OutA and OutB, respectively. Figure 6 illustrates the selection of output registers based on the values of OutASel and OutBSel control inputs.
We assigned the enables of 4 general purpose registers, R1, R2, R3 and R4, to 4-bit signal RegSel that selects the registers to apply the function that is determined by 3-bit FunSel, and in the same way, we assigned the enables of the scratch registers to ScrSel. Elaborated design of the system is given below (Figure 7).

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

Figure 6: OutASel and OutBSel controls

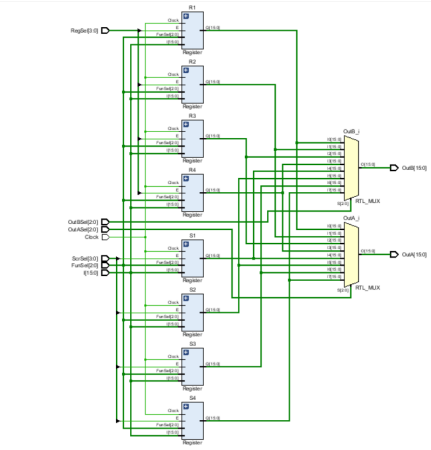


Figure 7: Verilog HDL Schematic of Register File

2.2.3 Address Register File:

In this subpart, we designed an address register file (ARF) system that contains three 16-bit address registers. One of those registers, Program Counter(PC), is a register in the processor of the computer that stores the address of the next instruction to be executed. The other one, address register (AR), is a component that holds memory addresses for read and write operations.[4] And the third one, stack pointer (SP) is a processor register that follows the top of the stack and is used to manage function calls and local variables. FunSel and RegSel work as in part 2b. Registers, inputs and outputs of Address Register File is shown Figure 8.

RegSel	Enable Address Registers
000	All address registers are enabled. (Function selected by FunSel will be applied to PC, AR, and SP.)
001	PC and AR are enabled. (Function selected by FunSel will be applied to PC and AR.)
010	PC and SP are enabled. (Function selected by FunSel will be applied to PC and SP.)
011	PC is enabled. (Function selected by FunSel will be applied to PC.)
100	AR and SP are enabled. (Function selected by FunSel will be applied to AR and SP.)
101	AR is enabled. (Function selected by FunSel will be applied to AR.)
110	SP is enabled. (Function selected by FunSel will be applied to SP.)
111	NO address register is enabled. (All registers retain their values.)

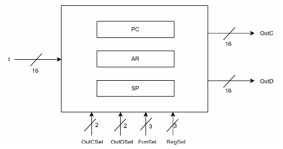


Figure 8: Registers, Inputs and Outputs of ARF

For implementation of this part, we assigned inputs, enable, outputs and clock signal to general purpose register's correspond inputs and outputs. Here, the enable inputs of PC, AR and SP were assigned in the form of their inverses according to the format requested from us, so we implemented it accordingly. The schematic of the design is as follows(Figure 9)

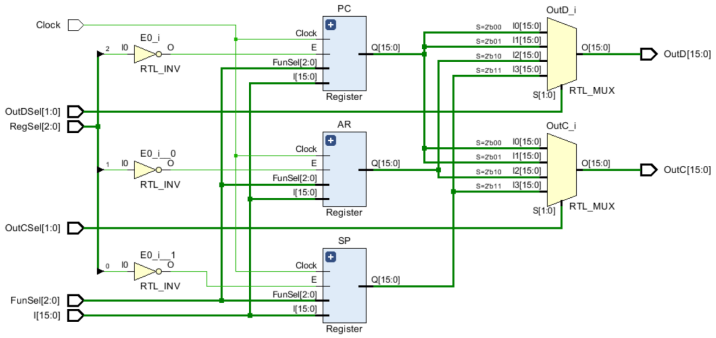
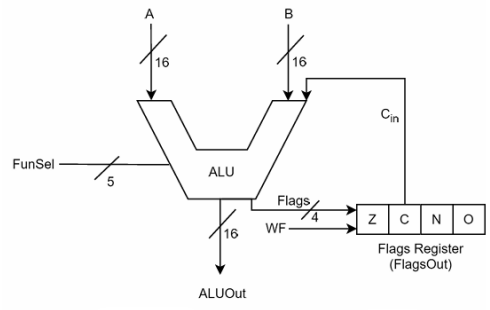


Figure 9: Verilog HDL Schematic of Address Register File

2.3 PART 3: ARITHMETIC LOGIC UNIT

In this part we have designed a arithmetic logic unit(ALU) with the capability to perform both arithmetic and logic operations. ALU has two 16-bit inputs, a 16-bit output and a 4-bit output for zero, carry,overflow and negative flags. We have assigned the flags to 4-bit FlagsOut register.ALU operations are not determined by clock signal,thus we

have implemented ALU operations in always@(*) block. Since flag update is controlled by both positive edge clock signal and Write Flag(WF) condition, we have implemented a separate always block to update flags. To be able to make operations with 8-bit inputs in 16-bit ALU, we have added 8-bit 0 to higher(15-8) half of ALU for convenience. Block diagram and characteristic table of the ALU (Figure 10) and elaborated design(Figure 11) is given below .



FunSel	ALUOut	Z	C	N	O
00000	A (8-bit)	+	-	+	-
00001	B (8-bit)	+	-	+	-
00010	NOT A (8-bit)	+	-	+	-
00011	NOT B (8-bit)	+	-	+	-
00100	A + B (8-bit)	+	+	+	+
00101	A + B + Carry (8-bit)	+	+	+	+
00110	A – B (8-bit)	+	+	+	+
00111	A AND B (8-bit)	+	-	+	-
01000	A OR B (8-bit)	+	-	+	-
01001	A XOR B (8-bit)	+	-	+	-
01010	A NAND B (8-bit)	+	-	+	-
01011	LSL A (8-bit)	+	+	+	-
01100	LSR A (8-bit)	+	+	+	-
01101	ASR A (8-bit)	+	+	-	-
01110	CSL A (8-bit)	+	+	+	-
01111	CSR A (8-bit)	+	+	+	-

FunSel	ALUOut	Z	C	N	O
10000	A (16-bit)	+	-	+	-
10001	B (16-bit)	+	-	+	-
10010	NOT A (16-bit)	+	-	+	-
10011	NOT B (16-bit)	+	-	+	-
10100	A + B (16-bit)	+	+	+	+
10101	A + B + Carry (16-bit)	+	+	+	+
10110	A – B (16-bit)	+	+	+	+
10111	A AND B (16-bit)	+	-	+	-
11000	A OR B (16-bit)	+	-	+	-
11001	A XOR B (16-bit)	+	-	+	-
11010	A NAND B (16-bit)	+	-	+	-
11011	LSL A (16-bit)	+	+	+	-
11100	LSR A (16-bit)	+	+	+	-
11101	ASR A (16-bit)	+	+	-	-
11110	CSL A (16-bit)	+	+	+	-
11111	CSR A (16-bit)	+	+	+	-

Figure 10: ALU Block Diagram and Operation Table

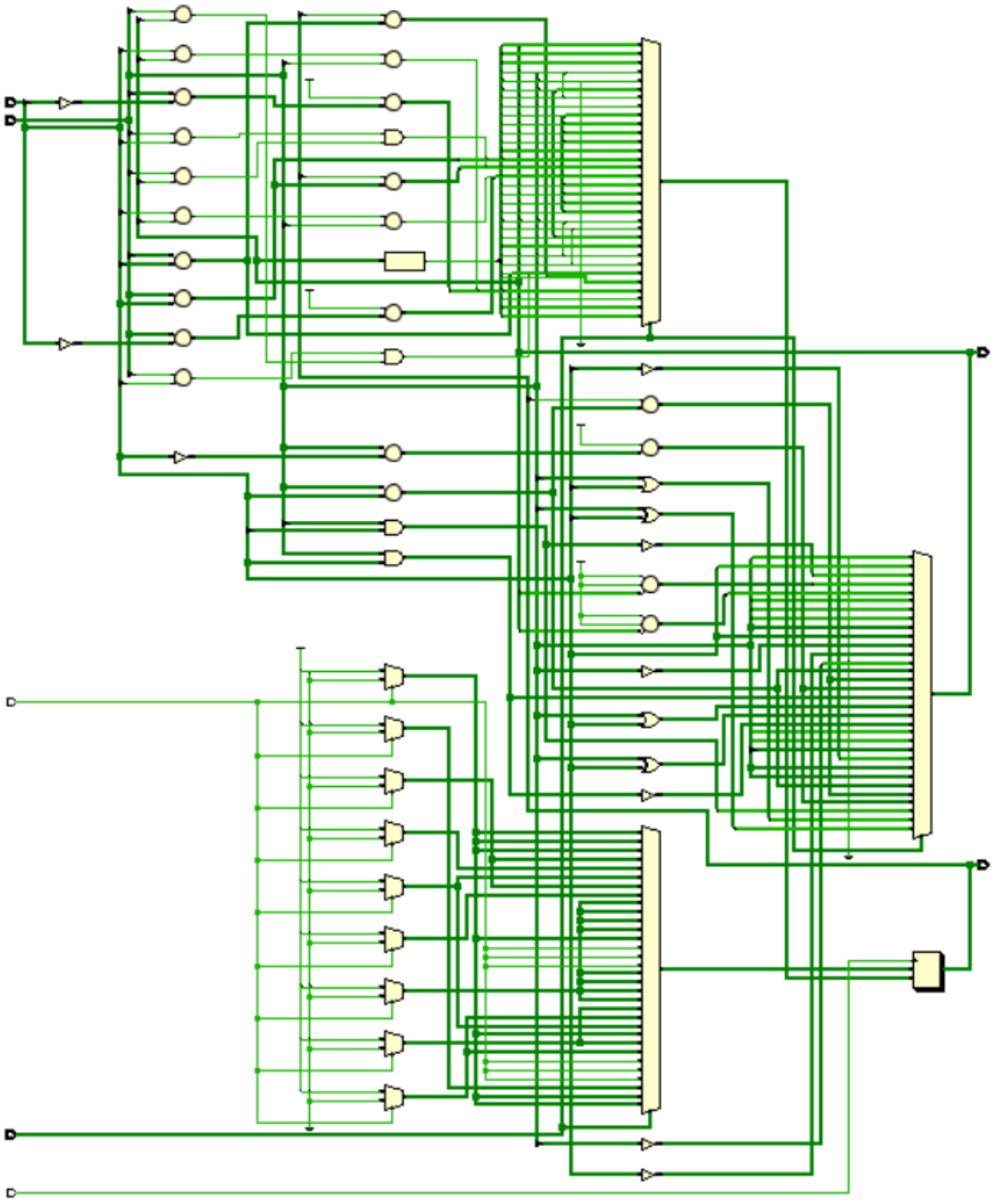
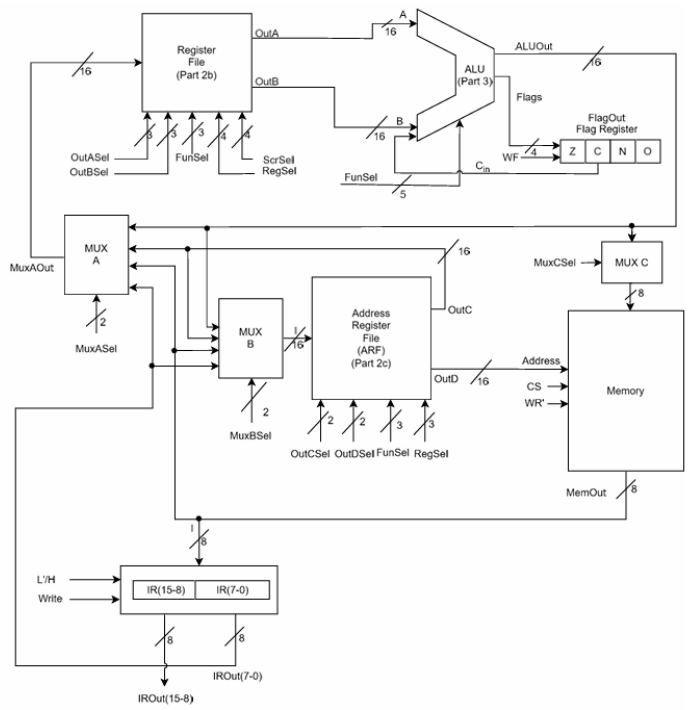


Figure 11: Verilog HDL Schematic of ALU

2.4 PART 4: ARITHMETIC LOGIC UNIT SYSTEM

In this part, we have designed intermediate wires between the parts that we have implemented in the previous parts, and connected them according the Figure 12 given below. These components form the basic architecture of a computer system, allowing it to execute instructions, manipulate data, and perform various computational tasks. Block diagram and multiplexers table (Figure 12) and elaborated design (Figure 13) follows.



MuxASel	MuxAOut
00	ALUOut
01	ARF OutC
10	Memory Output
11	IR (7:0)

MuxBSel	MuxBOut
00	ALUOut
01	ARF OutC
10	Memory Output
11	IR (7:0)

MuxCSel	MuxCOut
0	ALUOut(7-0)
1	ALUOut(15-8)

Figure 12: ALU System Block Diagram and Operation Table

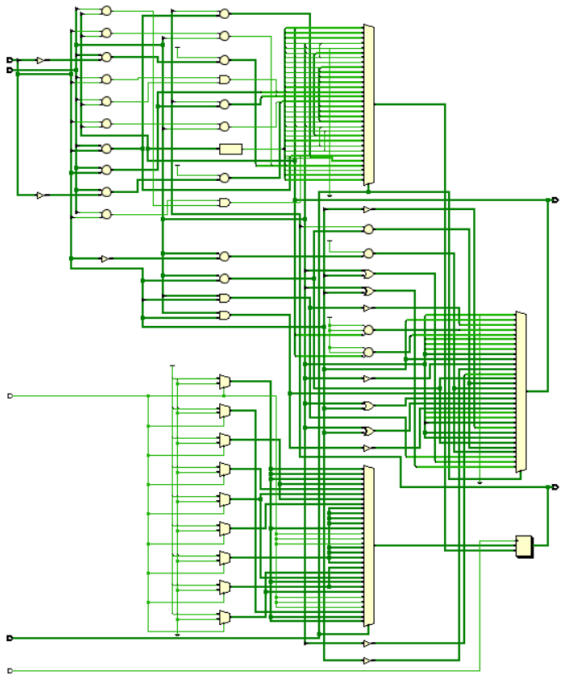


Figure 13: Verilog HDL Schematic of ALU

3 RESULTS

We have tested and simulated each part according to given simulation files. The results obtained are as follows.

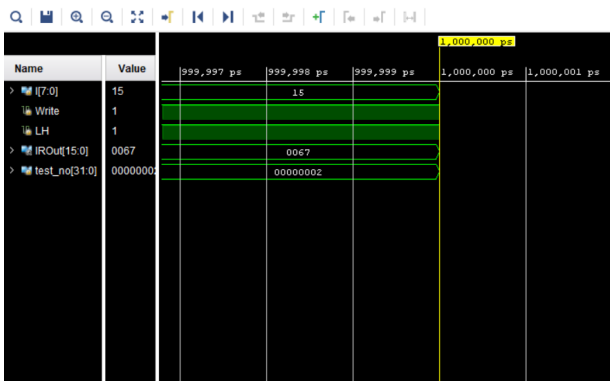


Figure 14: Simulation of Register

```
# run 1000ns
------------------------------------
Register Simulation Started
[PASS] Test No: 1, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 2, Component: Q, Actual Value: 0x0024, Expected Value: 0x0024
[PASS] Test No: 3, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 4, Component: Q, Actual Value: 0x0026, Expected Value: 0x0026
[PASS] Test No: 5, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 6, Component: Q, Actual Value: 0x0012, Expected Value: 0x0012
[PASS] Test No: 7, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 8, Component: Q, Actual Value: 0x0000, Expected Value: 0x0000
Register Simulation Finished
0 Test Failed
8 Test Passed
------------------------------------
```

Figure 15: Test Results of Register

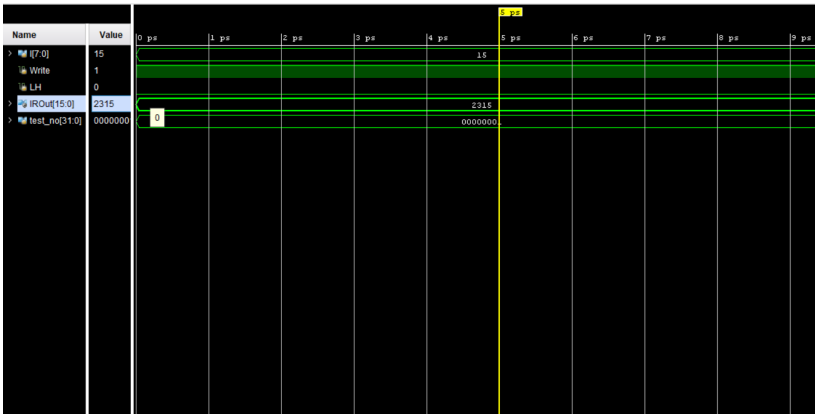


Figure 16: Simulation of Instruction Register(IR)

```
------------------------------------
InstructionRegister Simulation Started
[PASS] Test No: 1, Component: IROut, Actual Value: 0x2315, Expected Value: 0x2315
[PASS] Test No: 2, Component: IROut, Actual Value: 0x1567, Expected Value: 0x1567
InstructionRegister Simulation Finished
0 Test Failed
2 Test Passed
------------------------------------
```

Figure 17: Test Results of Instruction Register(IR)

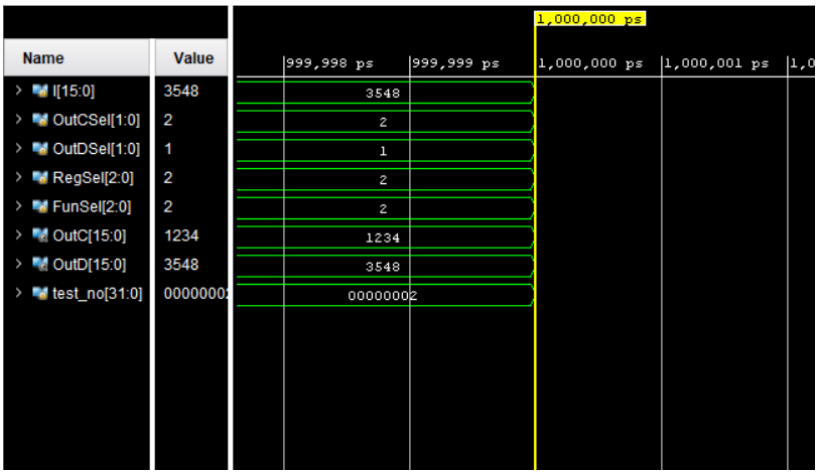


Figure 18: Simulation of Address Register File(ARF)

```
------------------------------------
AddressRegisterFile Simulation Started
[PASS] Test No: 1, Component: OutC, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutD, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
AddressRegisterFile Simulation Finished
0 Test Failed
4 Test Passed
```

Figure 19: Test Results of Address Register File (ARF)

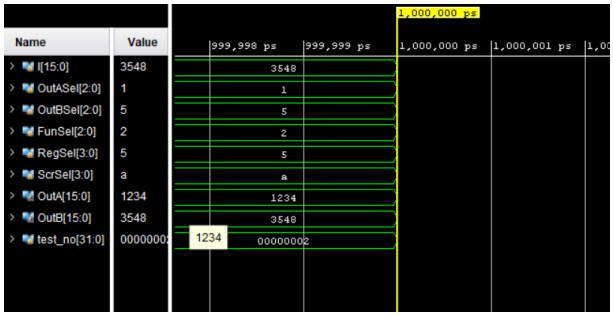


Figure 20: Simulation of Register File(RF)

```
-------------------------------------
RegisterFile Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutB, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
RegisterFile Simulation Finished
0 Test Failed
4 Test Passed
-------------------------------------
```

Figure 21: Test Results of Register File(RF)

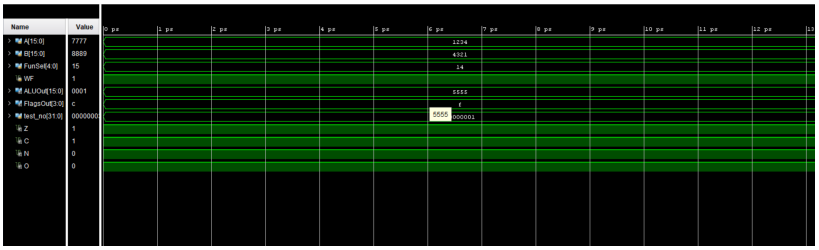


Figure 22: Simulation of ALU

```
ArithmeticLogicUnit Simulation Started
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 1, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: O, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: ALUOut, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
ArithmeticLogicUnit Simulation Finished
0 Test Failed
15 Test Passed
```

Figure 23: Test Results of ALU

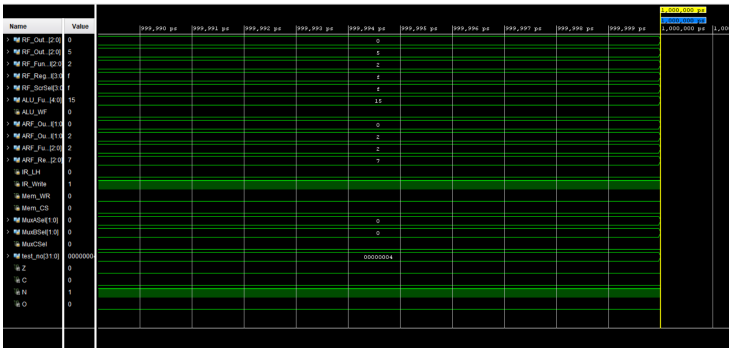


Figure 24: Simulation of ALU System

```
------------------------------------
ArithmeticLogicUnitSystem Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 1, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 1, Component: R2, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: S3, Actual Value: 0x0000, Expected Value: 0x0000
[FAIL] Test No: 2, Component: OutA, Actual Value: 0xfffe, Expected Value: 0x7777
[PASS] Test No: 2, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[FAIL] Test No: 2, Component: ALUOut, Actual Value: 0x8885, Expected Value: 0xfffe
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[FAIL] Test No: 2, Component: MuxAOut, Actual Value: 0x8885, Expected Value: 0xfffe
[FAIL] Test No: 2, Component: MuxBOut, Actual Value: 0x8885, Expected Value: 0xfffe
[FAIL] Test No: 2, Component: MuxCOut, Actual Value: 0x0085, Expected Value: 0x00fe
[PASS] Test No: 2, Component: R2, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: S3, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: PC, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 3, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 3, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 3, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[FAIL] Test No: 3, Component: IROut, Actual Value: 0x0015, Expected Value: 0x0000
[PASS] Test No: 4, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 4, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 4, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 4, Component: IROut, Actual Value: 0x0015, Expected Value: 0x0015
ArithmeticLogicUnitSystem Simulation Finished
6 Test Failed
27 Test Passed
------------------------------------
```

Figure 25: Test Results of ALU System

4 DISCUSSION

As we have explained in previous parts, we have started with describing different kind of registers and continued with implementing ALU which can perform both arithmetic (addition, substraction, complement etc.) and logic (AND, OR, XOR etc.) operations. After connecting the parts, we obtained an ALU system which is a basic architecture of a computer system, able to execute instructions, manipulate data, and perform various computational tasks.

5 CONCLUSION

Initially, we struggled to adapt to Verilog syntax and operational principles. At this stage, we benefited from the provided slides. We did not encounter much difficulty while implementing because instructions were given from basic to complex. However, during operations with 8-bit inputs and a 16-bit ALU, we faced uncertainty regarding whether to convert inputs to 16 bits with sign extension. Following the testing phase, we resolved this by appending 8-bit zeros to the higher half of the ALU to ensure successful test passage. Another problem that we encountered was ALU System simulation. We have failed 6 out of 33 tests but we could not solve them, problems were not connected systemically, our implementation passed all tests until the last part. After discussion with another groups we have realized that they were passing the tests that we have failed with same actual value obtained. In addition we could not tested with run.bat file due to "ECHO IS OFF" error, while echo was on. We have learned how to use Verilog HDL, basic computer architecture, how to connect parts, how to test and simulate implementations, how to handle with errors.

REFERENCES

[1] Digital Circuit Course Slide-1-2.

[2] Morris Mano. Computer system architecture third edition. *Textbook*.

[3] https://www.geeksforgeeks.org/instruction register/.

[4] https://www.geeksforgeeks.org/different-classes-of-cpu registers/.